Paper M02: Data MADs - Managing the Storage Domain Trio

Version: 1.3 Draft **Date**: October 17, 2025 **Status**: Draft - Awaiting Review

---

Abstract

This paper examines the Data domain within the Joshua ecosystem, focusing on three specialized MADs that manage distinct storage types through the resource-manager pattern. The structured data MAD manages relational databases for transactional and structured information. The semi-structured data MAD manages document stores for flexible schema data and conversation archives. The unstructured data MAD manages file systems and distributed storage for arbitrary binary data. Each MAD serves as a domain manager providing expertise in resource optimization, health monitoring, and lifecycle management, while the underlying storage resources remain separate infrastructure components. This pattern demonstrates how the Cellular Monolith Architecture separates domain expertise from resource provision, enabling intelligent storage management without tightly coupling MADs to specific storage technologies.

Keywords: data management, resource-manager pattern, relational databases, document stores, file systems, domain expertise

---

1. Introduction

1.1 Three Types of Storage

Digital systems require three fundamentally different storage types, each optimized for different data characteristics.

Structured storage handles data with fixed schemas and relational constraints. Tables, rows, columns, foreign keys, and transactions define this domain. SQL databases excel at enforcing data integrity, maintaining consistency, and enabling complex queries across related entities. Financial transactions, user accounts, and system configurations naturally fit structured storage.

Semi-structured storage handles data with flexible schemas and hierarchical organization. Documents, nested objects, and dynamic fields define this domain. Document databases excel at storing complex nested structures without requiring schema migrations for every field addition. Conversation histories, JSON configurations, and evolving data models naturally fit semi-structured storage.

Unstructured storage handles arbitrary binary data without inherent organization. Files, blobs, and streams define this domain. File systems excel at storing large objects, managing versions, and organizing through directories and metadata. Images, videos, documents, and binary artifacts naturally fit unstructured storage.

Systems that conflate these storage types create impedance mismatches. Storing files as database BLOBs wastes relational capabilities and creates performance issues. Storing structured transactional data in document stores loses integrity guarantees. Forcing relational structure onto flexible document data creates maintenance burden.

1.2 The Resource-Manager Pattern

The Joshua ecosystem implements storage through the resource-manager pattern: storage resources provide capacity and durability while MADs provide domain expertise and management intelligence.

Resources are the storage systems themselves—PostgreSQL databases, MongoDB instances, ZFS file systems, NAS appliances. They provide raw capabilities: storing data durably, retrieving data reliably, enforcing configured constraints. Resources are passive infrastructure components that respond to commands but don't make autonomous decisions.

Domain Managers are MADs specialized in particular storage domains. They understand optimal usage patterns, monitor resource health, optimize performance, manage lifecycles, and provide high-level abstractions to other MADs. Domain managers make intelligent decisions about how resources should be used.

This separation enables flexibility. Storage resources can be replaced or upgraded without changing domain manager interfaces. Domain managers can optimize resource usage based on workload patterns without requiring changes to underlying storage systems. The pattern scales independently—more resources scale capacity, more intelligent domain managers scale decision-making quality.

1.3 Three Domain Managers

The Data domain implements three specialized domain managers, each focused on a specific storage type.

The **structured data MAD** manages relational database resources, providing expertise in schema design, transaction management, query optimization, and data integrity. Other MADs interact with this component when they need transactional storage with referential integrity.

The **semi-structured data MAD** manages document database resources, providing expertise in document modeling, index design, flexible schema evolution, and conversation archival. Other MADs interact with this component when they need hierarchical storage with dynamic schemas.

The **unstructured data MAD** manages file system resources, providing expertise in file organization, versioning, distributed storage, and lifecycle management. Other MADs interact with this component when they need to store and retrieve large binary objects.

Each domain manager shares the same MAD pattern architecture—Thought Engine for reasoning, Action Engine for execution, Progressive Cognitive Pipeline for optimization. They differ only in domain expertise and the resources they manage.

1.4 Empirical Validation

The Data MAD architecture described in this paper has been empirically validated through two case studies:

V0 Architecture (Paper C01 / Appendix A): The 52-document Cellular Monolith generation leveraged semi-structured storage (Dewey conversation archive) and unstructured storage (Horace file catalog) to manage conversation history, document versions, and deliverable artifacts. This validated the resource-manager pattern at prototype scale, demonstrating that storage domain managers can efficiently coordinate resources during intensive multi-LLM operations. **See Appendix A for complete case study details.**

V1 Architecture (Paper C02 / Appendix B): The Synergos autonomous creation process relied on all three Data MADs—structured storage for task state, semi-structured storage for conversation archives, and unstructured storage for source code files. This validated the trio's ability to support complex software creation workflows while maintaining data integrity across concurrent operations. **See Appendix B for complete case study details.**

Together, these case studies provide empirical evidence that the three-domain Data architecture operates as designed, successfully managing diverse storage requirements through specialized domain expertise.

---

2. Structured Data Management

2.1 The Relational Domain

Structured data management centers on relational databases where data integrity and consistency are paramount. The structured data MAD provides expertise in this domain, managing PostgreSQL resources for the Joshua ecosystem.

Named after E.F. Codd, inventor of the relational model, this MAD understands relational theory, normalization principles, transaction semantics, and SQL optimization. It doesn't just execute database commands—it provides intelligent database administration through understanding of optimal patterns and anti-patterns.

2.2 Schema Management

Database schemas define the structure of relational data—tables, columns, constraints, indexes, relationships. The structured data MAD manages schema evolution as system requirements change.

When a new persistent MAD requires structured storage, it describes its data needs conversationally to the structured data MAD. The conversation covers entity types, attributes, relationships, access patterns, and integrity requirements. The structured data MAD designs an appropriate schema considering normalization, performance trade-offs, and future extensibility.

Schema migrations occur when existing MADs need schema changes. The structured data MAD analyzes proposed changes for backward compatibility, data preservation, and performance impact. It generates migration scripts that transform existing data safely, validates migrations in test environments, and coordinates deployment to minimize disruption.

This conversational schema management eliminates the traditional DBA bottleneck. MADs don't submit formal schema change requests through ticketing systems. They converse with the structured data MAD, which understands both the technical implications and the requesting MAD's operational context.

2.3 Transaction Management

Transactions ensure data consistency when multiple operations must succeed or fail atomically. The structured data MAD provides transaction coordination for operations spanning multiple MADs.

When a MAD begins an operation requiring transactional semantics, it creates a conversation with the structured data MAD describing the transactional boundaries. The structured data MAD ensures appropriate isolation levels, coordinates distributed transactions if multiple databases are involved, and handles rollback scenarios when errors occur.

This expertise extends to understanding transaction patterns. The structured data MAD recognizes when operations could cause deadlocks, suggests alternative sequencing to reduce contention, and optimizes transaction scope to minimize lock duration. These optimizations emerge from domain expertise rather than requiring requesting MADs to understand low-level transaction mechanics.

2.4 Query Optimization

Database performance depends heavily on query optimization—how queries are structured, which indexes exist, how the query planner interprets operations. The structured data MAD provides query optimization expertise.

When MADs describe data retrieval needs, the structured data MAD analyzes access patterns to recommend appropriate indexes. It monitors query performance through PostgreSQL's statistics, identifying slow queries and suggesting optimizations. It understands when to denormalize for performance, when materialized views make sense, and when query restructuring improves execution plans.

This optimization happens proactively. The structured data MAD monitors database workload continuously, identifying performance degradation before it impacts operations. It can suggest schema refinements based on observed access patterns, recommend index additions for frequently filtered columns, and identify opportunities for partitioning large tables.

2.5 Health Monitoring and Backup

Database health requires continuous monitoring of various metrics—connection pool saturation, disk space consumption, replication lag, lock contention. The structured data MAD monitors PostgreSQL health comprehensively.

When metrics indicate issues, the structured data MAD takes appropriate action. High connection pool usage triggers investigation of connection leaks. Disk space approaching capacity triggers data archival or storage expansion. Replication lag triggers investigation of bottlenecks in replication pipelines.

Backup management ensures data durability beyond database replication. The structured data MAD coordinates regular backup schedules, validates backup integrity through test restores, manages backup retention policies, and coordinates disaster recovery procedures when needed. This backup expertise includes understanding tradeoffs between backup frequency, restore time, and storage consumption.

---

3. Semi-Structured Data Management

3.1 The Document Domain

Semi-structured data management centers on document databases where schema flexibility and hierarchical organization are valuable. The semi-structured data MAD provides expertise in this domain, managing MongoDB resources for conversation archives and dynamic configuration.

This MAD understands document modeling, nested structures, dynamic schemas, and indexing strategies for document queries. It manages MongoDB instances ensuring efficient document storage and retrieval while accommodating schema evolution without migration overhead.

3.2 Conversation Archival

The conversation bus generates continuous streams of messages requiring permanent archival. Conversations contain variable structures—prose descriptions, structured JSON, embedded data, nested responses. This variability makes document storage ideal, specifically MongoDB for its flexible schema and rich query capabilities.

The semi-structured data MAD manages conversation archival in MongoDB by designing document schemas that accommodate conversation diversity while enabling efficient querying. Each conversation becomes a document with nested messages. Message metadata enables temporal queries, participant filtering, and topic-based retrieval.

Indexing strategies balance write performance against query flexibility. Conversations are written frequently but queried selectively. The semi-structured data MAD designs indexes supporting common query patterns—retrieve conversations by participant, find conversations in date ranges, search conversation content—without creating excessive write overhead.

Archival lifecycle management ensures conversation history remains accessible without consuming unlimited storage. The semi-structured data MAD implements retention policies, archiving older conversations to cheaper storage tiers while maintaining recent conversations in high-performance stores. Queries transparently access both tiers as needed.

3.3 Dynamic Configuration Storage

System configuration often requires schema flexibility. New configuration parameters appear as features are added. Configuration structures evolve as architectural patterns change. Document storage accommodates this evolution naturally.

The semi-structured data MAD manages configuration storage by providing MADs with flexible configuration spaces. Each MAD has a configuration document containing its parameters. When MADs need new configuration fields, they simply add them to documents without schema migrations.

This flexibility extends to nested configuration structures. A MAD might configure complex workflows, multi-stage pipelines, or hierarchical resource allocations. Document storage naturally represents these nested structures without requiring separate relational tables and foreign keys.

Validation ensures flexibility doesn't compromise correctness. The semi-structured data MAD can enforce validation rules on configuration documents, ensuring required fields exist, value ranges are respected, and referenced entities are valid. This provides safety without sacrificing flexibility.

3.4 Document Lifecycle Management

Documents have lifecycles—creation, active use, archival, eventual deletion. The semi-structured data MAD manages these lifecycles intelligently.

Recently created documents receive high-performance storage and frequent backups. Actively accessed documents remain in memory caches and fast storage tiers. Infrequently accessed documents migrate to cheaper storage with reduced backup frequency. Obsolete documents are eventually deleted according to retention policies.

This lifecycle management operates transparently. MADs interact with the semi-structured data MAD through consistent interfaces regardless of which storage tier currently holds documents. The semi-structured data MAD handles retrieval from appropriate tiers, migration between tiers based on access patterns, and lifecycle transitions based on age and retention policies.

3.5 Query Pattern Optimization

Document databases support flexible querying but require thoughtful index design for performance. The semi-structured data MAD optimizes for observed query patterns.

By monitoring queries, the semi-structured data MAD identifies common patterns—frequently filtered fields, common sorting criteria, typical aggregation operations. It creates indexes supporting these patterns without requiring requesters to specify index needs explicitly.

This optimization includes understanding tradeoffs. Indexes accelerate reads but slow writes. The semi-structured data MAD balances these tradeoffs based on workload characteristics. Write-heavy collections receive minimal indexing. Read-heavy collections receive comprehensive indexes supporting diverse query patterns.

---

4. Unstructured Data Management

4.1 The File Domain

Unstructured data management centers on file systems where arbitrary binary data requires storage, organization, and retrieval. The unstructured data MAD provides expertise in this domain, managing NAS, ZFS pools, and distributed file systems for the Joshua ecosystem.

This MAD understands file organization patterns, version control concepts, distributed storage architectures, and data lifecycle management. It makes file storage invisible to other MADs by handling all complexity internally while providing simple conversational interfaces for storing and retrieving files.

4.2 File Organization

Files require organization for efficient retrieval. Simple flat directories become unmanageable at scale. Hierarchical organization helps but requires consistent patterns. The unstructured data MAD implements intelligent file organization.

When MADs store files, they describe the file's purpose and relationships conversationally rather than specifying exact paths. The unstructured data MAD determines appropriate organization based on file type, creating MAD, retention requirements, and access patterns. It generates consistent paths, creates necessary directory structures, and manages organization evolution as patterns change.

This abstraction means MADs never manipulate paths directly. They request file storage by describing what the file is. They retrieve files by describing what they need. The unstructured data MAD translates these conversational requests into appropriate file system operations, handling all path management internally.

Metadata enriches file organization beyond hierarchical paths. The unstructured data MAD maintains metadata catalogs describing file contents, origins, relationships, and access history. This enables querying files by characteristics rather than remembering exact paths. A MAD can request "the most recent architectural diagram for the database schema" and receive the appropriate file based on metadata rather than requiring knowledge of the file system structure.

4.3 Version Control

Files evolve through revisions. Documents are edited, code is modified, configurations are adjusted. The unstructured data MAD provides version control for all stored files.

When a MAD updates a file, the unstructured data MAD preserves the previous version automatically. Version history enables retrieval of earlier revisions, comparison between versions, and rollback when needed. This versioning operates transparently—MADs store files normally, and versioning happens automatically.

Version retention balances storage consumption against historical value. Recent versions are preserved comprehensively. Older versions are thinned based on retention policies—perhaps keeping monthly snapshots beyond a certain age. The unstructured data MAD manages these retention decisions based on file importance, access patterns, and storage capacity.

This version control integrates with change tracking. When files are modified, the unstructured data MAD records context—which MAD made the change, why it was made, what conversation triggered the modification. This audit trail provides accountability and understanding of file evolution.

4.4 Distributed Storage Management

File storage scales through distribution across multiple physical resources. The unstructured data MAD manages distributed storage transparently, presenting a unified namespace while spreading data across multiple storage systems.

When files are stored, the unstructured data MAD determines appropriate physical locations based on various factors—file size, access frequency, geographic locality requirements, redundancy needs. Large files might be distributed across multiple storage nodes. Frequently accessed files might be replicated to multiple locations for faster retrieval. Critical files might receive geographic distribution for disaster recovery.

This distribution operates transparently. MADs store and retrieve files through conversational interaction without awareness of physical distribution. The unstructured data MAD handles all complexity—splitting large files across nodes, maintaining consistency replicas, routing retrieval requests to optimal locations, rebalancing storage as capacity changes.

Health monitoring ensures distributed storage remains reliable. The unstructured data MAD monitors storage node health, detects failures, redistributes data from failed nodes to healthy nodes, and maintains redundancy levels despite hardware failures. This self-healing capability prevents data loss from individual component failures.

4.5 Lifecycle Management

Files have lifecycles similar to documents but with additional considerations for large binary data. The unstructured data MAD manages these lifecycles to optimize storage cost and performance.

Newly created files receive fast storage and frequent backups. Actively accessed files remain on high-performance storage tiers. Infrequently accessed files migrate to cheaper, slower storage. Obsolete files are eventually deleted according to retention policies.

This lifecycle management considers file characteristics. Large multimedia files might migrate to archive storage more quickly than small configuration files. Version history might have different lifecycle policies than current versions. The unstructured data MAD applies appropriate policies based on file type, importance, and access patterns.

Integration with backup systems ensures durability throughout lifecycles. Recent files receive frequent backups to fast recovery storage. Archived files receive less frequent backups to cheaper media. The unstructured data MAD coordinates backup schedules, validates backup integrity, and manages recovery procedures when needed.

---

5. Domain Manager Coordination

5.1 Cross-Domain Operations

Some operations span multiple storage domains. Building a complete system snapshot requires coordinating relational data export, document collection archival, and file system backup. The three domain managers coordinate through conversational interaction.

When cross-domain operations are needed, initiating MADs create conversations including relevant domain managers. A backup operation conversation includes all three data MADs. Each contributes domain-specific expertise—the structured data MAD ensures consistent database snapshots, the semi-structured data MAD archives conversation collections, the unstructured data MAD backs up file systems. Coordination emerges from conversation without requiring explicit orchestration.

This conversational coordination enables flexible operation sequences. Simple backups might proceed in parallel across domains. Complex operations might require sequencing—perhaps database snapshots before file backups to ensure referential consistency. The domain managers determine appropriate coordination through conversational interaction based on operation requirements.

5.2 Consistent Abstractions

While managing different storage types, the three domain managers present consistent abstractions to other MADs. Requesting storage follows similar conversational patterns regardless of storage type. Retrieval uses similar interaction models. Lifecycle management follows comparable patterns.

This consistency emerges from shared MAD architecture. All three domain managers implement the same Thought Engine and Action Engine pattern. They participate in conversations through the same bus. They use similar reasoning patterns for different domain-specific decisions. The underlying storage technologies differ dramatically, but the conversational interfaces remain consistent.

Consistency extends to error handling and health reporting. When storage operations fail, domain managers report problems through similar conversational patterns. When resource health degrades, alerts follow comparable formats. This consistency enables other MADs to interact with data storage naturally without learning radically different patterns for each storage type.

5.3 Resource Abstraction

A key benefit of the resource-manager pattern: MADs interact with domain managers, not storage resources directly. This abstraction enables technology evolution without disrupting the ecosystem.

If PostgreSQL is replaced with a different relational database, only the structured data MAD requires changes. Its conversational interface remains constant—other MADs continue requesting structured storage the same way. The structured data MAD translates these requests into appropriate commands for the new database technology.

This abstraction extends to resource scaling. Adding storage capacity doesn't require changes to requesting MADs. Domain managers coordinate with infrastructure MADs to provision additional resources, distribute

load across expanded capacity, and maintain consistent interfaces despite infrastructure changes.

Geographic distribution operates similarly. If the ecosystem expands to multiple regions, domain managers handle replication and geographic routing transparently. MADs continue storing and retrieving data through the same conversational interfaces while domain managers ensure data is physically located appropriately for performance and compliance requirements.

---

6. Progressive Cognitive Pipeline Integration

6.1 Learning Storage Patterns

As domain managers operate, they learn common storage patterns through the Progressive Cognitive Pipeline. The LPPM observes repeated operations and compiles them into optimized processes.

When MADs repeatedly request similar data structures, the LPPM in the structured data MAD learns the pattern. Future similar requests execute faster through compiled process models rather than full reasoning cycles. This learning applies across all three domain managers—document storage patterns, file organization patterns, schema design patterns all become learnable processes.

The CET optimizes context assembly for storage decisions. Which conversation history is most relevant for understanding schema requirements? Which previous storage decisions inform current optimization choices? The CET learns to assemble optimal context for each domain manager's decision-making.

The DTR routes deterministic storage operations reflexively. Simple file retrieval by known identifier doesn't require full reasoning—it routes directly to execution. Complex storage design decisions require full cognitive processing. The DTR learns these routing patterns from operational history.

6.2 Cross-Domain Learning

Because all three domain managers share the same PCA architecture, learning in one domain can inform others. Lifecycle management patterns learned by the unstructured data MAD might inform document lifecycle decisions. Index optimization approaches from the semi-structured data MAD might suggest analogous optimizations for the structured data MAD.

This cross-domain learning happens through the conversation bus. Domain managers observe each other's conversations and decision patterns. When relevant patterns emerge, they adapt those patterns to their own domains. The shared architecture makes this learning natural—the learning mechanisms are identical, only the domain-specific training data differs.

6.3 Self-Optimization

Domain managers can request their own performance optimization from the meta-programming component. When the structured data MAD identifies query performance issues that architectural changes could address, it describes the need conversationally. The meta-programming component composes a database optimization team. The team implements enhancements. The structured data MAD gains improved capabilities.

This self-optimization creates continuously improving storage management. Domain managers don't just manage storage—they continuously improve how they manage storage by requesting their own enhancement when they identify opportunities.

---

7. Current Implementation Status

For complete implementation status and version progression details, see Paper J02: System Evolution and Current State.

7.1 Structured Data MAD

The structured data MAD is operational at V1 managing PostgreSQL databases for system state and structured application data. Schema management works conversationally for new MAD storage requirements. Transaction coordination handles single-database transactions reliably. Query monitoring identifies slow queries for optimization.

Areas for enhancement include comprehensive query optimization automation, distributed transaction coordination across multiple databases, and advanced backup orchestration with automated integrity validation.

7.2 Semi-Structured Data MAD

The semi-structured data MAD is operational at V1 managing MongoDB instances for conversation histories and dynamic configuration. Document schema design handles conversation archival effectively. Index management supports common query patterns. Configuration storage provides flexible schema evolution.

Areas for enhancement include automated lifecycle management with intelligent tier migration, cross-collection query optimization, and real-time document change streaming to interested MADs.

7.3 Unstructured Data MAD

The unstructured data MAD is operational at V1 managing local file systems with NAS integration. File organization uses consistent patterns. Metadata cataloging enables content-based retrieval. Basic version control preserves file history.

Areas for enhancement include comprehensive distributed storage management across multiple geographic regions, automated lifecycle policies based on access patterns and content analysis, and integration with cloud storage providers for hybrid storage architectures.

---

8. Conclusion

The Data domain demonstrates how the resource-manager pattern separates domain expertise from resource provision. Three specialized MADs manage three storage types—structured relational data, semi-structured documents, and unstructured files—by providing intelligent management of underlying storage resources.

This separation enables flexibility in storage technology choices, consistent abstractions across different storage types, and continuous optimization through the Progressive Cognitive Pipeline. Domain managers evolve independently from storage resources, learning improved management patterns over time while maintaining stable conversational interfaces.

The three-way division reflects fundamental differences in data characteristics. Structured data requires integrity and consistency. Semi-structured data requires flexibility and hierarchy. Unstructured data requires capacity and organization. Specialized domain managers for each type provide appropriate expertise without requiring other MADs to understand low-level storage mechanics.

The Data domain embodies the Cellular Monolith principle of shared architecture with specialized function. All three domain managers share the same MAD pattern and cognitive architecture. They differ only in domain expertise and the resources they manage. This architectural consistency enables natural coordination, cross-domain learning, and unified evolution while maintaining clear specialization boundaries.

References

1. Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks"
2. MongoDB document database architecture and patterns
3. ZFS filesystem concepts and distributed storage patterns

4. Paper J03: Cellular Monolith Architecture (resource-manager pattern)
5. Paper J04: Progressive Cognitive Pipeline (learning patterns)

Paper M02 - Draft v1.3 - October 17, 2025